International Journal of Modern Engineering Research (IJMER)
www.ijmer.com Vol.2, Issue.4, July-Aug. 2012 pp-2409-2416 ISSN: 2249-6645

Transaction Processing In Replicated Data in the DDBMS

Ashish Srivastava[1], Udai Shankar[2], Sanjay Kumar Tiwari[3]
*(Department Deptt. of Computer Science & Engineering, Madan Mohan Malaviya Engineering College, Gorakhpur).

ABSTRACT: *The transactional Processing in Replicated Data for distributed system has been around for many years and it is considered a well-established and mature technology. The conventional transaction model, although suitable for predictable database applications such as banking and airline reservation systems, does not provide much flexibility and high performance when used for complex applications such as object oriented systems, long-lived transactions or distributed systems. In this paper we describe the transaction-processing model of distributed database includes data, Transaction, Data Manager, and Transaction Manager and their transaction process. We will study about concurrency problem of sequence of synchronization techniques for transaction with respect to distributed database. The spirit of discussion in a decomposition of the concurrency control problem into two major sub problems: read-write and write-write synchronization. We describe a sequence of synchronization techniques for solving each sub complexity.*

Keywords: *Real time system, Replication, Distributed Database, Distributed processing, Transaction, Transaction Manager, two-phase commit, Concurrency Control, Synchronization.*

I. Introduction

A real-time system is one that must process information and produce a response within a specified time, else risk severe consequences, including failure. That is, in a system with a real-time constraint it is no good to have the correct action or the correct answer after a certain deadline: it is either by the deadline or it is useless. Database replication based on group communication systems has been proposed as an efficient and flexible solution for data replication. Replicated is the key characteristic in improving the availability of data distributed systems. Replicated data is stored at multiple server sites so that it can be accessed by the user even when some of the copies are not available due to server/site failures.[1] A Major restriction to using replication is that replicated copies must behave like a single copy, i.e. mutual consistency as well internal consistency must be preserved, Synchronization techniques for replicated data in distributed database systems have been studied in order to increase the degree of consistency and to reduce the possibility of transaction rollback. [2]

In replicated database systems, copies of the data items can be stored at multiple sites. The potential of data replication for high data availability and improved read performance is crucial to RTDBS. In contrast, data replication introduces its own problems. Access to a data item is no longer control exclusively by a single site; instead, the access control is distributed across the sites each storing the copy of the data item. It is necessary to ensure that mutual consistency of the replicated data is provided

Distributed data base system is a technique that is used to solve a single problem in a heterogeneous computer network system. A major issue in building a distributed database system is the transactions atomicity. When a transaction runs across into two sites, it may happen that one site may commit and other one may fail due to an inconsistent state of transaction. Two-phase commit protocol is widely used to solve these problems. The choice of commit protocol is an important design decision for distributed database system. A commit protocol in a distributed database transaction should uniformly commit to ensure that all the participating sites agree to the final outcome and the result may be either a commit or an abort situation. Many real times database applications are distributed in nature [3] these include the aircraft control, stock trading, network management, factory automation etc.

II. DISTRIBUTED DATABASE SYSTEMS (DDBS)

A distributed database is a database that is under the control of a central database management system (DBMS) in which storage devices are not all attached to a common CPU. It may be stored in multiple computers located in the same physical location, or may be dispersed over a network of interconnected computers. Collections of data can be distributed across multiple physical locations. Distributed database system (DDBS) is system that has distributed data and replicated over several locations. Data may be replicated over a network using horizontal and vertical fragmentation similar to projection and selection operations in Structured Query Language (SQL). The database shares the problems of access control and transaction management, such as user concurrent access control and deadlock detection and resolution. On the other hand, however, DDBS must also cope with different problems. Accessing of data control and transaction management in DDBS needs different methods to monitor data access and update to distributed and replicated databases. Distributed database systems (DDBS) are systems that have their data distributed and replicated over several locations; unlike the centralized data base system (CDBS), where one copy of the data is stored. Data may be replicated over a network using horizontal and vertical fragmentation similar to projection and selection operations in Structured Query Language (SQL). Both types of database share the same problems of access control and transaction management, such as user concurrent access control and deadlock detection and resolution. On the other hand however, DDBS must also cope with different problems. Access control and transaction management in DDBS require different rules to monitor data retrieval and update to distributed and replicated databases [4, 5].Oracle, as a leading Database Management Systems (DBMS) employs the two-phase commit technique to maintain a consistent state for the databases [6]. The objective of this paper is to

explain transaction management in DDBMS and how to implements this technique. To assist in understanding this process, an example is given in the last section. It is hoped that this understanding will encourage organizations to use and academics to discuss DDBS and to successfully capitalize on this feature of Database. The next section presents advantages, disadvantages, and failures in Distributed Database Systems. Subsequent sections provide discussions on the fundamentals of transaction management, two-phase commit, homogenous distributed database system implementation of the two-phase commit, and, finally, an example on how the two phases commit works.

2.1Advantages of Distributed Database system (DDBS)

Since organizations tend to be geographically dispersed, a DDBS fits the organizational structure better than traditional centralized DBS. Improved Availability-A failure does not make the entire system inoperable and Improved Reliability-Data may be replicatedEach location will have its local data as well as the ability to get needed data from other locations via a communication network. Moreover, the failure of one of the servers at one site won't render the distributed database system inaccessible. The affected site will be the only one directly involved with that failed server. In addition, if any data is required from a site exhibiting a failure, such data may be retrieved from other locations containing the replicated data [7]. The performance of the system will improve, since several machines take care of distributing the load of the CPU and the I/O. Also, the expansion of the distributed system is relatively easy, since adding a new location doesn't affect the existing ones.

2.2 Disadvantages of Distributed DBS

On the other hand, DDBS has several disadvantages. A distributed system usually exhibits more complexity and cost more than a centralized one. Security-network must be made secure Integrity Control More Difficult This is true because the hardware and software involved need to maintain a reliable and an efficient system. All the replication and data retrieval from all sites should be transparent to the user. The cost of maintaining the system is considerable since technicians and experts are required at every site. Another main disadvantage of distributed database systems is the issue of security. Handling security across several locations is more complicated. In addition, the communication between sites may be tapped to.

2.3 Issues in Distributed Database Design

We have to consider three key issues in distributed database design

- Data Allocation: where are data placed? Data should be stored at site with "optimal" distribution.
- Fragmentation: relation may be divided into a number of sub-relations (called fragments),which are stored in different sites.
- Replication: copy of fragment may be maintained at several sites.

III. FUNDAMENTALS OF TRANSACTION

Transaction deals with the problems of keeping the database in a consistent state even when concurrent accesses and failures occur.

3.1 What is a Transaction

A transaction consists of a series of operations performed on a database. The important issue in transaction management is that if a database was in a consistent state prior to the initiation of a transaction, then the database should return to a consistent state after the transaction is completed. This should be done irrespective of the fact that transactions were successfully executed simultaneously or there were failures during the execution,[8]. A transaction is a sequence of operations that takes the database from a consistent state to another consistent state. It represents a complete and correct computation. Two types of transactions are allowed in our environment: query transactions and update transactions. Query transactions consist only of read operations that access data objects and return their values to the user. Thus, query transactions do not modify the database state. Two transactions conflict if the read-set of one transaction intersects with the write-set of the other transaction. During the voting process, Update transactions consist of both read and write operations. Transactions have their time-stamps constructed by adding 1 to the greater of either the current time or the highest time-stamp of their base variables.Thus; a transaction is a unit of consistency and reliability. The properties of transactions will be discussed later in the properties section. Each transaction has to terminate. The outcome of the termination depends on the success or failure of the transaction. When a transaction starts executing, it may terminate with one of two possibilities:

1. The transaction aborts if a failure occurred during its execution
2. The transaction commits if it was completed successfully.

Example of a transaction that aborts during process 2 (P2). On the other hand, an example of a transaction that commits, since all of its processes are successfully completed [9, 10].

3.2Properties of Transactions

A Transaction has four properties that lead to the consistency and reliability of a distributed data base. These are Atomicity, Consistency, Isolation, and Durability [6].

ACID property of transaction: The concept of a database transaction (or atomic transaction) has evolved in order to enable both a well-understood database system behavior in a faulty environment where crashes can happen any time, and recovery from a crash to a well understood database state. A database transaction is a unit of work, typically encapsulating a number of operations over a database (e.g., reading a database object, writing, acquiring lock, etc.), an abstraction supported in database and also other systems. Each transaction has well defined boundaries in terms of which program/code executions are included in that transaction (determined by the transaction's programmer via special transaction commands). Every database transaction obeys the following rules (by support in the database system; i.e., a database system is designed to guarantee them for the transactions it runs):

Atomicity: This refers to the fact that a transaction is treated as a unit of operation. Consequently, it dictates that either all the actions related to a transaction are completed or none of them is carried out. For example, in the case of a crash, the system should complete the remainder of the transaction, or it will undo all the actions pertaining to this transaction. The recovery of the transaction is split into two types corresponding to the two types of failures: Atomicity means

that users do not have to worry about the effect of incomplete transactions. Transactions can fail for several kinds of reasons:

1. Hardware failure: A disk drive fails, preventing some of the transaction's database changes from taking effect.
2. System failure: The user loses their connection to the application before providing all necessary information.
3. Database failure: E.g., the database runs out of room to hold additional data.
4. Application failure: The application attempts to post data that violates a rule that the database itself enforces such as attempting to insert a duplicate value in a column.

Consistency: Every transaction must leave the database in a consistent (correct) state, i.e., maintain the predetermined integrity rules of the database (constraints upon and among the database's objects). A transaction must transform a database from one consistent state to another consistent state (however, it is the responsibility of the transaction's programmer to make sure that the transaction itself is correct, i.e., performs correctly what it intends to perform (from the application's point of view) while the predefined integrity rules are enforced by the DBMS). Thus since a database can be normally changed only by transactions, all the database's states are consistent. An aborted transaction does not change the database state it has started from, as if it never existed (atomicity above).

Isolation: According to this property, each transaction should see a consistent database at all times. Consequently, no other transaction can read or modify data that is being modified by another transaction. If this property is not maintained, one of two things could happen to the data base.
a. Lost Updates: this occurs when another transaction (T2) updates the same data being modified by the first transaction (T1) in such a manner that T2 reads the value prior to the writing of T1 thus creating the problem of loosing this update.
b. Cascading Aborts: this problem occurs when the first transaction (T1) aborts, then the transactions that had read or modified data that has been used by T1 will also abort.

Durability: Durability is the DBMS's guarantee that once the user has been notified of a transaction's success the transaction will not be lost, the transaction's data changes will survive system failure, and that all integrity constraints have been satisfied, so the DBMS won't need to reverse the transaction. Many DBMSs implement durability by writing transactions into a transaction log that can be reprocessed to recreate the system state right before any later failure. A transaction is deemed committed only after it is entered in the log. Durability does not imply a permanent state of the database. A subsequent transaction may modify data changed by a prior transaction without violating the durability principle .The concept of atomic transaction has been extended during the years to what has become a Business transaction, which actually implement types of Workflow and are not atomic. However also such enhanced transactions typically utilize atomic transactions as components [11, 12].

3.3 Type of distributed transaction

By structure, distributed transaction is dividing into two types. A flat transaction, FT, is an operation, performed on a database, which may consist of several simple actions. From the client's point of view the operation must be executed indivisibly. Main disadvantage with FTs that if one action fails the whole transaction must abort. Issues related to distributed transaction: There are a number of issues or problems, which are peculiar to a distributed database and these, require novel solutions. These include the following:

3.3.1Distributed query optimisation: In a distributed database the optimisation of queries by the DBMS itself is critical to the efficient performance of the overall system. Query optimisation must take into account the extra communication costs of moving data from site to site, but can use whatever replicated copies of data are closest, to execute a query. Thus it is a more complex operation than query optimisation in centralised databases.

3.3.2Distributed update propagation: Update propagation in a distributed database is problematic because of the fact that there may be more than one copy of a piece of data because of replication, and data may be split up because of partitioning. Any updates to data performed by any user must be propagated to all copies throughout the database. The use of snapshots is one technique for implementing this.

3.3.3Distributed catalog management: The distributed database catalog entries must specify site(s) at which data is being stored in addition to data in a system catalog in a centralised DBMS. Because of data partitioning and replication, this extra information is needed. There are a number of approaches to implementing a distributed database catalog. Centralized- Keep one master copy of the catalog, fully replicated Keep one copy of the catalog at each site, Partitioned -Partition and replicate the catalog as usage patterns demand, Centralised/partitioned- Combination of the above.

3.3.4Distributed concurrency control: Concurrency Control4 in distributed databases can be done in several ways. Locking and timestamping are two techniques, which can be used, but timestamping is generally preferred. The problems of concurrency control in a distributed DBMS are more severe than in a centralised DBMS because of the fact that data may be replicated and partitioned. If a user wants unique access to a piece of data, for example to perform an update or a read, the DBMS must be able to guarantee unique access to that data, which is difficult if there are copies throughout the sites in the distributed database.
A number of problems arise while dealing with concurrency control and recovery issues in distributed databases. Some of the major problems are:
Site failure: There are situation when one or more sites in a DDBMS fail. In such situations, consistency and integrity of the database must be restored.
Network Problems: When communication network fails, causing one or more sites to be cut off from the rest of the sites in the DDBMS environment
Data Duplication: Multiple copies of the database must be monitor carefully for maintaining consistency.
Distributed Transaction: A problem arise when a transaction distributed across various sites. Some of the sites are successfully committing/rolling, while the others may not be successfully done.

Distributed Deadlocks: In DDBMS, a deadlock may occur in any one or many sites. So, careful handling is necessary.

3.3.5Transaction Concurrency: If transactions are executed serially, i.e., sequentially with no overlap in time, no transaction concurrency4 exists. However, if concurrent transactions with interleaving operations are allowed in an uncontrolled manner, some unexpected, undesirable result may occur. Here

IV. Transaction Processing In Replicated Data in the DDBMS

A transaction is a logical unit of work constituted by one or more SQL statements executed by a single user. A transaction begins with the user's first executable SQL statement and ends when it is committed or rolled back by that user. A remote transaction contains only statements that access a single remote node. A distributed transaction contains statements that access more than one node. A distributed transaction is a transaction that includes one or more statements that, individually or as a group, update data on two or more distinct nodes of a distributed database. The term replication refers to the operation of copying and maintaining database objects in multiple databases belonging to a distributed system. The terms distributed database system and database replication are related, yet distinct. In a pure (that is, not replicated) distributed database, the system manages a single copy of all data and supporting database objects. Typically, distributed database applications use distributed transactions to access both local and remote data and modify the global database in real-time. While replication relies on distributed database technology, database replication offers applications benefits that are not possible within a pure distributed database environment. Most commonly, replication is used to improve local database performance and protect the availability of applications because alternate data access options exist.[13,14,16] For example, an application may normally access a local database rather than a remote server to minimize network traffic and achieve maximum performance. Furthermore, the application can continue to function if the local server experiences a failure, but other servers with replicated data remain accessible. A new component, which is a replication manager module, has been recently added to the system, in order to maintain replicated data.

4.1Transaction-Processing Model:

A DDBMS contains four components: transactions (T), Transaction Manager (TMR), Data Manager (DMR), and data (D). Transactions communicate with TMRs, TMRs communicate with DMRs, and DMRs manage the D. TMRs supervise transactions. Each transaction executed in the DDBMS is supervised by a single TMR, meaning that the transaction issues all of its database operations to that TMR. Any distributed computation that is needed to execute the transaction is managed by the TMR. Four operations are defined at the transaction-TMR interface.

READ (A): returns the value of A (a logical data item) in the current logical database state.

WRITE (A, new-value): creates a new logical database state in which A has the specified new value.

BEGIN and END operations to bracket transaction executions.

DMRs manage the stored database, functioning as backend database processors. In response to commands from transactions, TMRs issue commands to DMRs specifying stored data items to be read or written.

In a centralized DBMS, private workspaces are part of the Transaction Manager (TMR) and data can freely move between a transaction and its workspace, and between a workspace and the Data Manager (DMR). Whereas in a DDBMS TMRs and DMRs may run at different sites and the movement of data between a TM and a DM can be expensive. To reduce this cost, many DDBMSs employ query optimization procedures which regulate the flow of data between sites. How a Transaction (T) reads and writes data in these workspaces is a query optimization problem and has no direct effect on concurrency control. Suppose T is updating x, y, z stored at DMRx, DMRy, DMRz, and suppose T's TMR fails after issuing DMR- write(x), but before issuing the dm-writes for y and z. At this point the database is incorrect. However, in a DDBMS, other TMRs remain operational and can access the incorrect database. To avoid this problem, prewrite commands must be modified slightly. In addition to specifying data items to be copied onto secure storage, prewrites also specify which other DMRs are involved in the commitment activity. Then if the TMR fails during the second phase of two-phase commit, the DMRs whose dm-writes were not issued can recognize the situation and consult the other DMRs involved in the commitment. If any DMR received a dmr-write, the remaining ones act as if they had also received the command. In a DDBMS these are processed as follows.

BEGIN: The TMR creates a private work space for T.

READ (A): The TMR checks T's private workspace to see if a copy of A is present. If so, that copy's value is made available to T. Otherwise the TMR selects some stored copy of A, say xi, and issues read(x,) to the DMR at which x, is stored. The DMR responds by retrieving the stored value of x, from the database, placing it in the private workspace. The TMR returns this value to T.

WRITE (A, new-value): The value of A in T's private workspace is updated to newvalue, assuming the workspace contains a copy of A. Otherwise; a copy of A with the new value is created in the workspace.

END: Two-phase commit begins.

For each A updated by T, and for each stored copy x, of A, the TMR issues a prewrite (x,) to the DMR that stores x,. The DMR responds by copying the value of A from T's private workspace onto secure storage internal to the DMR. After all prewrites are processed, the TMR issues dm-writes for all copies of all logical data items updated by T.

A DMR responds to dmr-write(x,) by copying the value of x, from secure storage into the stored database. After all dmr-writes are installed, T's execution is finished.

4.2 SYNCHRONIZATION TECHNIQUES BASED ON TWO-PHASE LOCKING

Two-phase locking (2PL) synchronizes reads and writes by explicitly detecting and preventing conflicts between concurrent operations. Earlier than reading data item x, a transaction must "own" a read lock on x. Before writing into x, it must "own" a write lock on x. The ownership of locks is governed by two rules:

(1) Different transactions cannot simultaneously own conflicting locks

(2) Once a transaction surrenders ownership of a lock, it may never obtain additional locks.

The definition of conflicting lock depends on the type of synchronization being performed:

For 'rw' synchronization two locks conflict if

(a) Both are locks on the same data item, and

(b) One is a read lock and the other is a write lock;

for 'ww' synchronization two locks conflict if

(a) Both are locks on the same data item, and

(b) Both are write locks.

The second lock ownership rule causes every transaction to obtain locks in a two- phase manner. During the growing phase the transaction obtains locks without releasing any locks. By releasing a lock the transaction enters the shrinking phase. During this phase the transaction releases locks, and, by rule 2, is prohibited from obtaining additional locks. When the transaction terminates (or aborts), all remaining locks are automatically released. A common variation is to require that transactions obtain all locks before begin- ning their main execution. This variation is called predeclaration. Some systems also require that transactions hold all locks until termination.

4.2.1Performance of 2PL

A performance of 2PL amounts to building a 2PL scheduler, a software module that receives lock requests and lock releases and processes them according to the 2PL specification. The basic way to implement 2PL in a distributed database is to distribute the schedulers along with the database, placing the scheduler for data item x at the DMR were x is stored. In this implementation read locks may be implicitly requested by dmr reads and write locks may be implicitly requested by prewrites. If the requested lock cannot be granted, the operation is placed on a waiting queue for the desired data item. Write locks are implicitly released by dmr-writes. However, to release readlocks, special lockrelease operations are required. When a lock is released, the operations on the waiting queue of that data item are processed first-in/first-out (FIFO) order. However, if a transaction updates A, then it must update all copies of A, and so must obtain write locks on all copies of A.

4.2.2Primary Copy 2PL

Primary copy 2PL is a 2PL technique that pays attention to data redundancy. One copy of each logical data item is designated the primary copy; before accessing any copy of the logical data item, the appropriate lock must be obtained on the primary copy. For read locks this technique requires more communication than basic 2PL. Suppose xl is the primary copy of logical data item A, and suppose transaction T wishes to read some other copy, x,, of A. To read x, T must communicate with two DMRs, the DMR where As is stored.. But under basic 2PL, T would only communicate with x,'s DMR. For write locks, however, primary copy 2PL does not incur extra communication.

4.2.3Voting 2PL Method

Voting 2PL is another performance of 2PL that exploits data redundancy. Voting 2PL is derived from the majority consensus technique of Thomas and is only suitable for 'ww 'synchronization. To understand voting, we must examine it in the context of two-phase commit. Suppose transaction T wants to write into A. Its TMR sends prewrites to each DMR holding a copy of A. For the voting protocol, the DMR always responds immediately. It acknowledges receipt of the prewrite and says "lock set" or "lock blocked." After the TM receives acknowledgments from the DMRs, it counts the number of "lockset" responses: if the number constitutes a majority, then the TMR behaves as if all locks were set. Otherwise, it waits for "lockset" operations from DMRs that originally said "lock blocked." Deadlocks aside, it will eventually receive enough "lockset" operations to proceed. Since only one transaction can hold a majority of locks on A at a time, only one transaction writing into A can be in its second commit phase at any time [17, 18]. All copies of A thereby have the same sequence of writes applied to them. transaction's locked point occurs when it has obtained a majority of its write locks on each data item in its write set. When updating many data items, a transaction must obtain a majority of locks on every data item before it issues any dmr-writes. In principle, voting 2PL could be adapted for 'rw' synchronization. Before reading any copy of a transaction requests read locks on all copies of A; when a majority of locks are set, the transaction may read any copy. This technique works but is overly strong: Correctness only requires that a single copy of A be locked--namely, the copy that is read--yet this technique requests locks on all copies. For this reason we deem voting 2PL to be inappropriate for rw synchronization.

V. Two-Phase Commit of transaction in Distributed database System

In transaction processing, databases, and computer networking, the two-phase commit protocol (2PC) is a type of atomic commitment protocol (ACP). It is a distributed algorithm that coordinates all the processes that participate in a distributed atomic transaction on whether to commit or abort (roll back) the transaction (it is a specialized type of consensus protocol). The protocol achieves its goal even in many cases of temporary system failure (involving process, network node, communication, etc. failures), and is thus widely utilized[17,18].However, it is not resilient to all possible failure configurations, and in rare cases user (e.g., a system's administrator) intervention is needed to remedy outcome. To accommodate recovery from failure (automatic in most cases) the protocol's participants use logging of the protocol's states. Log records, which are typically slow to generate but survive failures, are used by the protocol's recovery procedures. Many protocol variants exist that primarily differ in logging strategies and recovery mechanisms. Though usually intended to be used infrequently, recovery procedures comprise a substantial portion of the protocol, due to many possible failure scenarios to be considered and supported by the protocol. In a "normal execution" of any single distributed transaction, i.e., when no failure occurs, which is typically the most frequent situation, the protocol comprises two phases:

1. The commit-request phase (or voting phase), in which a coordinator process attempts to prepare all the transaction's participating processes (named participants, cohorts, or workers) to take the necessary steps for either committing or aborting the transaction and to vote, either "Yes": commit (if the transaction participant's local portion execution has ended properly), or "No": abort (if a problem has been detected with the local portion).

2. The commit phase, in which, based on voting of the cohorts, the coordinator decides whether to commit (only if all have voted "Yes") or abort the transaction (otherwise), and notifies the result to all the cohorts. The cohorts then follow with the needed actions (commit or abort) with their local transactional resources (also called recoverable resources; e.g., database data) and their respective portions in the transaction's other output (if applicable).

5.1Commit request phase

1. The coordinator sends a query to commit message to all cohorts and waits until it has received a reply from all cohorts.
2. The cohorts execute the transaction up to the point where they will be asked to commit. They each write an entry to their undo log and an entry to their redo log.
3. Each cohort replies with an agreement message (cohort votes Yes to commit), if the cohort's actions succeeded, or an abort message (cohort votes No, not to commit), if the cohort experiences a failure that will make it impossible to commit.

5.2Commit phase

Success: If the coordinator received an agreement message from all cohorts during the commit-request phase:

1. The coordinator sends a commit message to all the cohorts.
2. Each cohort completes the operation, and releases all the locks and resources held during the transaction.
3. Each cohort sends an acknowledgement to the coordinator.
4. The coordinator undoes the transaction when all acknowledgements have been received

Failure:

If any cohort votes No during the commit-request phase (or the coordinator's timeout expires):

1. The coordinator sends a rollback message to all the cohorts.
2. Each cohort undoes the transaction using the undo log, and releases the resources and locks held during the transaction.
3. Each cohort sends an acknowledgement to the coordinator.
4. The coordinator undoes the transaction when all acknowledgements have been received.

The protocol proceeds in two phases, namely the prepare and the commit phase, which explains the protocol's name. The protocol is executed by a coordinator process, while the participating servers are called participants. When the transaction's initiator issues a request to commit the transaction, the coordinator starts the first phase of the 2PC protocol by querying—via prepare messages—all participants whether to abort or to commit the transaction. The master initiates the first phase of the protocol by sending PREPARE (to commit) messages in parallel to all the cohorts. Each cohort that is ready to commit first force-writes a prepare log record to its local stable storage and then sends a YES vote to the master. At this stage, the cohort has entered a prepared state wherein it cannot unilaterally commit or abort the transaction but has to wait for the final decision from the master. On the other hand, each cohort that decides to abort force-writes an abort log record and sends a NO vote to the master. Since a NO vote acts like a veto, the cohort is permitted to unilaterally abort the transaction without waiting for a response from the master.

After the master receives the votes from all the cohorts, it initiates the second phase of the protocol. If all the votes are YES, it moves to a committing state by force writing a commit log record and sending COMMIT messages to all the cohorts. Each cohort after receiving a COMMIT message moves to the committing state, force-writes a commit log record, and sends an ACK message to the master. If the master receives even one NO vote, it moves to the aborting state by force-writing an abort log record and sends ABORT messages to those cohorts that are in the prepared state. These cohorts, after receiving the ABORT message, move to the aborting state, force write an abort log record and send an ACK message to the master. Finally, the master, after receiving acknowledgements from all the prepared cohorts, writes an end log record and then "forgets" the transaction. The 2PC may be carried out with one of the following methods: Centralized 2PC, Linear 2PC, and Distributed 2PC, [17, 18].

5.3The Centralized Two-Phase Commit Protocol

In the Centralized 2PC communication is done through the coordinator's process only, and thus no communication between subordinates is allowed. The coordinator is responsible for transmitting the PREPARE message to the subordinates, and, when the votes of all the subordinates are received and evaluated, the coordinator decides on the course of action: either abort or COMMIT. This method has two phases:

1. First Phase: In this phase, when a user wants to COMMIT a transaction, the coordinator issues a PREPARE message to all the subordinates, (Mohan et al., 1986). When a subordinate receives the PREPARE message, it writes a PREPARE log and, if that subordinate is willing to COMMIT, sends a YES VOTE, and enters the PREPARED state; or, it writes an abort record and, if that subordinate is not willing to COMMIT, sends a NO VOTE. A subordinate sending a NO VOTE doesn't need to enter a PREPARED state since it knows that the coordinator will issue an abort. In this case, the NO VOTE acts like a veto in the sense that only one NO VOTE is needed to abort the transaction. The following two rules apply to the coordinator's decision.

a. If even one participant votes to abort the transaction, the coordinator has to reach a global abort decision.
b. If all the participants vote to COMMIT, the coordinator has to reach a global COMMIT decision.

2. Second Phase: After the coordinator reaches a vote, it has to relay that vote to the subordinates. If the decision is COMMIT, then the coordinator moves into the committing state and sends a COMMIT message to all the subordinates informing them of the COMMIT. When the subordinates receive the COMMIT message, they, in turn, move to the committing state and send an acknowledge (ACK) message to the coordinator. When the coordinator receives the ACK messages, it ends the transaction. If, on the other hand, the coordinator reaches an ABORT decision, it sends an ABORT message to all the subordinates. Here, the coordinator doesn't need to send an ABORT message to the subordinate(s) that gave a NO VOTE.

5.4The Linear Two-Phase Commit Protocol

In the linear 2PC, subordinates can communicate with each other. The sites are labeled 1 to N, where the coordinator is numbered as site 1. Accordingly, the propagation of the PREPARE message is done serially. As such, the time

required to complete the transaction is longer than centralized or distributed methods. Finally, node N is the one that issues the Global COMMIT. The two phases are discussed below:

First Phase: The coordinator sends a PREPARE message to participant 2. If participant 2 is not willing to COMMIT, then it sends a VOTE ABORT (VA) to participant 3 and the transaction is aborted at this point. If participant 2, on the other hand, is willing to commit, it sends a VOTE COMMIT (VC) to participant 3 and enters a READY state. In turn, participant 3 sends its vote till node N is reached and issues its vote.

Second Phase: Node N issues either a GLOBAL ABORT (GA) or a GLOBAL COMMIT (GC) and sends it to node N-1. Subsequently, node N-1 will enter an ABORT or COMMIT state. In turn, node N-1 will send the GA or GC to node N-2, until the final vote to commit or abort reaches the coordinator, node

5.5 The Distributed Two-Phase Commit Protocol

In the distributed 2PC, all the nodes communicate with each other. According to this protocol, as Figure 5 shows, the second phase is not needed as in other 2PC methods. Moreover, each node must have a list of all the participating nodes in order to know that each node has sent in its vote. The distributed 2PC starts when the coordinator sends a PREPARE message to all the participating nodes. When each participant gets the PREPARE message, it sends its vote to all the other participants. As such, each node maintains a complete list of the participants in every transaction. Each participant has to wait and receive the vote from all other participants. When a node receives all the votes from all the participants, it can decide directly on COMMIT or abort. There is no need to start the second phase, since the coordinator does not have to consolidate all the votes in order to arrive at the final decision.

VI. DATABASE MANAGEMENT SYSTEM: THE TWO-PHASE COMMIT

A distributed database system is a network of two or more s Databases that reside on one or more machines. A distributed system that connects four databases. An application can simultaneously access or modify the data in several databases in a single distributed environment. For a client application, the location and platform of the databases are transparent. You can also create synonyms for remote objects in the distributed system so that users can access them with the same syntax as local objects. For example, if you are connected to database mfg but want to access data on database headquarters, creating a synonym on manufacturing for the remote dept table enables you to issue this query[18] ,The database is a distributed database management system, which employs the two-phase commit to achieve and maintain data reliability. The DB2 database is a distributed database management system, which employs the two-phase commit to achieve and maintain data reliability. The following sections explain DB2's two-phase implementation procedures. How Session maintains between nodes in each transaction, DB2 constructs a session tree for the participating nodes. The session tree describes the relations between the nodes participating in any given transaction. Each node plays one or more of the following roles:

6.1The Branch Tree

In each transaction, Oracle constructs a branch tree for the participating nodes. The session tree describes the relations between the nodes participating in any given transaction. Each node plays one or more of the following roles [10]:

6.1.1 Client(C): A client is a node that references data from another node.

6.1.2*. Database Server (DS*): A server is a node that is being referenced by another node because it has needed data. A database server is a server that supports a local database.

6.1.3. Global Coordinator (GC*):* The global coordinator is the node that initiated the transaction, and thus, is the root of the branch tree. The operations performed by the global coordinator are as follows:

- In its role as a global coordinator and the root of the branch tree, all the SQL statements, procedure calls, etc., are sent to the referenced nodes by the global coordinator. Instructs all the nodes, except the COMMIT point site, to PREPARE
- If all sites PREPARE successfully, then the global coordinator instructs the COMMIT point site to initiate the commit phase
- If one or more of the nodes send an abort message, then the global coordinator instructs all nodes to perform a rollback.

6.1.*4*. Local Coordinator: A local coordinator is a node that must reference data on another node in order to complete its part. The local coordinator carries out the following functions:

- Receiving and relaying status information among the local nodes
- Passing queries to those nodes
- Receiving queries from those nodes and passing them on to other nodes
- Returning the results of the queries to the nodes that initiated them.

6.1.5. Commit Point Site: Before a COMMIT point site can be designated, the COMMIT point strength of each node must be determined. The COMMIT point strength of each node of the distributed database system is defined when the initial connection is made between the nodes. The COMMIT point site has to be a reliable node because it has to take care of all the messages. When the global coordinator initiates a transaction, it checks the direct references to see which one is going to act as a COMMIT point site. The COMMIT point site cannot be a read-only site. If multiple nodes have the same COMMIT point strength, then the global coordinator selects one of them. In case of a rollback, the PREPARE and COMMIT phases are not needed and thus a COMMIT point site is not selected. A transaction is considered to be committed once the COMMIT point site commits locally.

***6.*2Two-Phase Commit and the Database Implementation**

The transaction manager of the homogenous Oracle8 database necessitates that the decision on what to do with a transaction to be unanimous by all nodes. This requires all concerned nodes to make one of two decisions: commit and complete the transaction, or abort and rollback the transaction. The Oracle engine automatically takes care of the commit

[19]. or rollback of all transactions, thus, maintaining the integrity of the database. The following will describe the two phases of the transaction manager.

6.2.1. PREPARE Phase (PP): The PP starts when a node, the initiator, asks all participants, except the commit point site, to PREPARE. In the PP, the requested nodes have to record enough information to enable them either to commit or abort the transaction. The node, after replying to the requestor that it has PREPARED, cannot unilaterally perform a COMMIT or abort. Moreover, the data that is tied with the COMMIT or abort is not available for other transactions.
Each node may reply with one of three responses to the initiator. These responses are defined below:
a. Prepared: the data has already been modified and that the node is ready to COMMIT. All resources affected by the transaction are locked.
b. Read-only: the data on the node has not been modified. With this reply, the node does not PREPARE and does not participate in the second phase.
c. Abort: the data on the node could not be modified and thus the node frees any locked resources for this transaction and sends an abort message to the node that referenced it.

6.2.2. COMMIT Phase (CP): Before the CP begins, all the referenced nodes need to have successfully PREPARED. The COMMIT phase begins by the global coordinator sending a message to all the nodes instructing them to COMMIT. Thus, the databases across all nodes are consistent.

VII. CONCLUSIONS

At the present time Transaction management is an fully grown thought in distributed data base management systems (DDBMS) for research area for research. In this paper, we have reviewed the basic concepts Transaction Processing In Replicated Data. Many associations do not implement distributed databases because of its difficulty. They simply resort to centralized databases. However, with global organizations and multi-tier network architectures, distributed implementation becomes a necessity. It is hoped that this paper to will assist organization in the implementation of distributed databases when installing homogenous DBMS, or give confidence organizations to journey from centralized to distributed DBMS.We talk about the basic concept of transaction in distributed database systems, and also discussed the advantage, property and operations transaction in distributed environments. It is really important for database to have the ACID properties to perform. We have presented the basics of distributed database technology as well as the techniques that help in distribution of database in transaction-processing model. Also, Discussion regarding the framework for the design and analysis of distributed database concurrency control algorithms. The framework has two main components are system model that provides common terminology and concepts for describing a variety of concurrency control algorithms, and a problem decomposition that decomposes concurrency control algorithms into readwrite and write-write synchronization subalgorithms. We have considered synchronization subalgorithms outside the context of specific concurrency control algorithms.

References

[1] R. Abbott and H. Garcia-Molin a, "Scheduling 'Real-Time Transactions: a Performance Evahration", F&c. of 14th VLDB Conj., August 1988.
[2] Sang Hyuk Son, "Replicated Data Management in Distributed Database Systems", ACM Simgod,, Vol. 17, Issue 4, Dec 1998, Newyork,USA, pp-62-69.
[3] Jayant. H, Carey M, Livney,1992, "Data Access Scheduling in Firm Real time Database Systems", Real Time systems Journal, 4
[4] M, Valduriez P, 1991, Principles of Distributed Database Systems, Prentice-Hall.P. Bernstein, V. Hadzilacos and N. Goodman,
[5] G. Coulouris, J. Dollimore, T. Kindberg: Disributed Systems, Concepts and Design, Addison–Wesley, 1994.
[6] Oracle8 Server Distributed Database Systems, Oracle, 3-1 – 3-35.
[7] Ozsu, Tamer M., and V.alduriez, Patrick [1991], Principles of Distributed Database Systems, Prentice
[8] Mohan, C.; Lindsay, B.; and Obermarck, R. [1986],"Transaction Management in the R* Distributed Database Management System." ACM Transactions on Database Systems, Vol. 11, No. 4, December1986, 379-395.
[9] S. Ceri, M.A.W. Houtsma, A.M. Keller, P. Samarati: A Classification of Update Methods for Replicated Databases, via Internet, May 5, 1994.
[10] D. Agrawal, A.El. Abbadi: The Tree Quorum Protocol: An Efficient Approach for Managing Replicated Data. in Proc. of VLDB Conf. pp 243-254, 1990.
[11] Distributed Transaction Processing on an Ordering Network By Rashmi Srinivasa, Craig Williams, Paul F. Reynolds (2002)
[12] Ghazi Alkhatib, Transaction Management in Distributed Database: the Case of Oracle's Two-Phase Commit, Vol. 13(2)
[13] E. Cooper, "Analysis of Distributed Commit Protocols", Proc. of ACM Sigmod Conj., June 1982.
[14] Ramamritham,Son S. H, and DiPippo L,2004, Real-Time Databases and Data Services, Real-Time Systems J., vol. 28, 179-216.
[15] Jayanta Singh and S.C Mehrotra et all, 2010,"Management of missed transaction in a distributed system through simulation", Proc. Of IEEE.
[16] Udai Shanker, "Some Performance Issues In Distributed Real Time Database System", PhD Thesis, December,2005
[17] D. Agrawal, A.J. Bernstein, P. Gupta, S. Sengupta, "Distributed Optimistic Concurrency Control with Reduced Rollback," Distributed Computing, vol. 2, no. 1, pp. 45-59, 1987.
[18] R. Agrawal, M.J. Carey and L.W. McVoy, "The Performance of Alternative Strategies for Dealing with Deadlocks in Database Management Systems," IEEE Trans. Software Eng., vol. 13, no. 12, pp. 1,348-1,363, Dec. 1987.
[19] Boutros B. S. and Desai B. C., "A Two-Phase Commit Protocol and its Performance," in Proceedings of the 7th International Workshop on Database and Expert Systems Applications, pp.100-105, 1996.